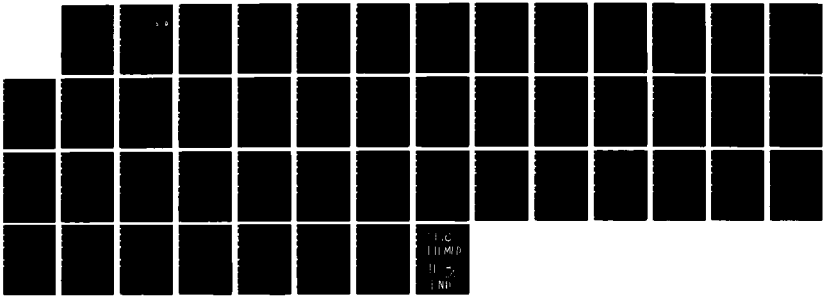
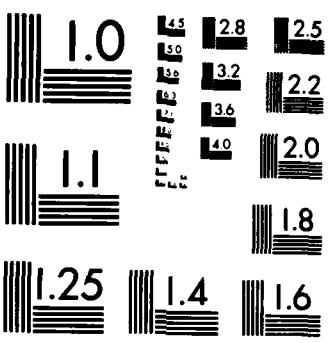
AD-A165 310 PROCESS AND DATA MANAGEMENT IN A RECONFIGURABLE DISTRIBUTED NETWORK(U) CALIFORNIA UNIV BERKELEY ELECTRONICS RESEARCH LAB C V RAMAMOORTHY 15 OCT 84 1/1

UNCLASSIFIED DASG60-81-C-0025 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

FINAL REPORT

PROCESS AND DATA MANAGEMENT IN A RECONFIGURABLE DISTRIBUTED NETWORK

Contract DASG60-81-C-0025
(2/27/81 - 10/15/84)

C. V. RAMAMOORTHY
Principal Investigator

SPONSORED BY

THE BALLISTIC MISSILE DEFENSE ADVANCED TECHNOLOGY CENTER

"The views, options, and/or findings contained in this report are those of the author and should not be construed as an official Department of the Army position policy, or decision, unless so designated by other official documentation."

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. ADA 165310	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) PROCESS AND DATA MANAGEMENT IN A RECONFIGURABLE DISTRIBUTED NETWORK		5. TYPE OF REPORT & PERIOD COVERED FINAL (2/27/81 - 10/15/84)
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) C. V. Ramamoorthy		8. CONTRACT OR GRANT NUMBER(s) DASG60-81-C-0025
9. PERFORMING ORGANIZATION NAME AND ADDRESS Electronics Research Laboratory University of California Berkeley, CA 94720		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ballistic Missile Defense Systems Command Department of the Army P.O. Box 1500 - Huntsville, AL 35807		12. REPORT DATE
		13. NUMBER OF PAGES 44
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES The views, options, and/or findings contained in this report are those of the author and should not be construed as an official Department of the Army position policy, or decision, unless so designated by other official documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number) communication protocols, distributed systems, global information, load balancing, process allocation, coordination of distributed computation, intelligent control, reconfigurable distributed system, recovery, multiple copy update, consensus problem, adaptive hierarchical routing, directory management

20. ABSTRACT (Continue on reverse side if necessary and identify by block number) Our objective in this research is to investigate certain unsolved problems in the design and management of distributed networks and thereby to develop design methods, analysis techniques and algorithms appropriate for such networks. The distinguishing features of the class of networks under consideration are the dynamic real-time processing loads as well as dynamic changes in the topology and connectivity of the networks and the unrealiable nature of the communication links.

(continued on other side)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(abstract continued)

The problems we are tackling include the management of global information, routing control, the design/analysis of communication protocols, process allocation, load balancing, coordination of distributed computation, and intelligent control.

(-A-)

ABSTRACT

Our objective in this research is to investigate certain unsolved problems in the design and management of distributed networks and thereby to develop design methods, analysis techniques and algorithms appropriate for such networks. The distinguishing features of the class of networks under consideration are the dynamic real-time processing loads as well as dynamic changes in the topology and connectivity of the networks and the unreliable nature of the communication links.

The problems we are tackling include the management of global information, routing control, the design/analysis of communication protocols, process allocation, load balancing, coordination of distributed computation, and intelligent control.

Table of Contents

1. INTRODUCTION

1.1. Overview and Objectives

Our objective is to investigate certain unsolved problems in the design and management of distributed networks and thereby to develop design methods, analysis techniques and algorithms appropriate for such networks. The distinguishing features of the class of networks under consideration are the dynamic changes in real-time system processing load as well as the dynamic changes in the topology and connectivity of the network and the unreliable nature of the communication links.

The problems we are tackling include the management of global information, routing control, the design/analysis of communication protocols, process allocation, load balancing, coordination of distributed computation, and intelligent control.

1.2. Research Motivations

A key function of a distributed network is to allow nodes to share information. We call such information global. Global information may be used for control purposes, e.g. node status, routing tables, etc. or it may be data used by multiple users, e.g. database files. However, the distinguishing features of dynamic networks mentioned above make the management of global information extremely difficult. The information must be transmitted to the recipient(s) reliably in the presence of malfunctioning/malicious nodes quite apart from the problem of connectivity. Consistency must be maintained in spite of concurrent accesses(the synchronization problem), replicas of information must be consistently and efficiently managed(the multiple update problem), the consistency must be restored if some operations do not complete(the recovery problem), the information must be distributed to conform to access patterns (the data allocation problem), the global information must be easily accessible to the users(naming and accessing problem), and the recipient of the global information must have some guarantee that the received information is up-to-date(the currency problem). We will examine these problems in Chapter 2.

In order to convey global information among nodes in the dynamic environment, we need routing control. As the network size grows larger and larger, both centralized and distributed conventional routing algorithms become less suitable. The difficulties come from the excessiveness of the communication and computation overhead involved. Instead, hierarchical schemes would have to be used. On the other hand, as the networks under study are dynamic and unreliable in nature, the routing schemes should also be adaptive and distributed. We will examine the routing problems in Chapter 3.

The exchange of global information must be performed in accordance with communication protocols. In order to maintain reliable and smooth operations in dynamic networks, several classes of communication protocols are needed. Configuration control protocols are used to report any topology and connectivity changes. Reliable end-to-end protocols are used to reliably and efficiently transmit network control information and user traffic. Channel access control protocols are necessary to effectively share the common channel among users. Therefore, efficient protocol analysis techniques and protocol synthesis methods are needed to ensure the correctness of the above mentioned protocols. We will examine the problems of protocol analysis and protocol synthesis in Chapter 4.

A distributed computation is usually defined as multiple processes working collectively towards a common goal. These processes interact with system objects distributed over the nodes, and they also communicate with each other. Overhead is incurred due to their consumption of system resources, such as processor time and communication bandwidth. In particular, if a lot of inter-node data communication is required, the communication overhead may impose a significant performance penalty on the distributed computation. We will address the problem of process allocation in a computer network environment and find a solution so that system resources can be more effectively utilized using our process allocation scheme. This is discussed in Chapter 5.

Besides the process allocation, we are also interested in the general problem of load-sharing which also addresses the problem of process

migration among network nodes. Load-sharing is a policy for redistributing the aggregate demand among the processors in the network so that the overall performance of the system improves. Optimal load-sharing requires foreknowledge of the runtime characteristics of all processes and is computationally expensive. An approach to a suboptimal solution is discussed in Chapter 6.

To support distributed computations, we must also consider the problem of coordination among nodes that participate in the computation. A computation may involve resources distributed all over the network and complex interactions among processes. We propose to develop a distributed *Make* program that is more powerful than the current version of *Make* program which could only take care of computations run on a local machine. The basic idea is to use a *Makefile* to specify the data dependence and control dependence of a particular distributed job and to have a distributed *Make* program to distribute the load of computation to different nodes to achieve high parallelism and handle the coordination among different nodes. Moreover, it would facilitate users in different nodes to develop software or document cooperatively. Combining with the process allocation strategy and process migration strategy, a distributed *Make* program would help distributed computations to achieve high resource utilization and the desired degree of fault tolerance for particular application. In the context of large dynamic networks, this would help save some precious computation results in a dynamically changing environment and satisfy the goal of real time processing. The related issues are discussed in Chapter 7.

Finally, we are also interested in the application of both Artificial Intelligence and Database technologies to a dynamic network environment to solve three major problems:

(a) Communication Subnet Interconnection Problem

(b) Query Processing Prolem

(c) Intelligent Control Problem

These problems will be addressed in Chapter 8.

In Appendix, we attached all the dissertations that have been published under this project.

2. GLOBAL INFORMATION MANAGEMENT

2.1. Introduction

Global Information is information which is derived from, and is relevant to all the nodes in a distributed system/computer network. It is used for various applications, with status maintenance and consistency of data and processors, being among those of prime concern. With the increased importance of distributed systems and networks, the need for global information cannot be overemphasized. When we talk of information which can be derived only from all the nodes in a network, we immediately run into difficulties. One of the main ones is that any single node has but a partial view of the whole system and hence can by itself make no decisions that are of global consequence. The system has to be *fault tolerant* in lieu of the fact that some of the nodes could stop functioning; and the system has to be failure resilient to a certain degree too. Real-time systems for critical applications should be able to manage computer resources in such a manner so as to be fault tolerant in the presence of hardware and software failures. The following issues have to be tackled by *Global Information Management*.

2.1.1. Collection and Dissemination

The possible existence of malfunctioning components, i.e. failed links, crashed nodes, network partitions or relaying of incorrect information, makes the collecting and distributing of information from one or more nodes a difficult task.

2.1.2. Synchronization

When we are trying to access global information by means of concurrently executing processes, we ought to maintain a high degree of concurrency in the access itself. This leads to synchronization problems. The various processes accessing an item of data have to be what in database terminology is known as *serializable*. This essentially means that for some particular interleaved execution of these concurrent processes there exists some strictly serial schedule of execution which achieves exactly the same

effect. Various methods using locking, timestamps, tickets, tokens, etc. have been used to solve this problem in centralized and distributed databases. The same problem arises in the context of operating systems when shared data structures like I/O service queues and buffers, memory buffer pointers, files, etc. have to be manipulated. However, shared data structures are more difficult to maintain in distributed systems.

2.1.3. Currency

A change in global data cannot have its effect propagated to every relevant site instantaneously. There is some finite delay involved, which is dependent upon the characteristics of the network and the algorithms used. Computations made on the basis of old data can have unprecedented, and often undesired and sometimes even unpleasant consequences. The degree of unpleasantness is a function of the application at hand. For example, a query *decomposing algorithm* which proceeds on the basis of old information can be recovered quite simply. On the other hand several distributed deadlock detection algorithms have been known to fail in the presence of race conditions, which results from the non-zero delay in the network. Actual deadlocks may be hidden while spurious ones spring up.

2.1.4. Recovery

On talking of recovery, we always think in terms of having some state information upon which we can rely. In case of failures or crashes we can go back and restart from this reliable state. This brings us to the concept of *atomic actions*, or what in database terminology is called as a transaction. A consistent state is defined as being a state in which some invariant assertions on the global information are satisfied. An atomic action is one which takes the system from one consistent state to another consistent state. These atomic actions are not actually so, and hence they may abort (due to software or hardware failures) when the system is not in a consistent state. In this case it is the duty of the recovery manager to restore the system to a consistent state, usually the one just before the start of the current atomic action. The concept of atomic actions has been extended to

give rise to the idea of *recovery blocks* intended to provide error detection and recovery for both hardware and software failures. A recovery block has a *primary* alternate, zero or more *secondary* alternates, and an *acceptance test*. The primary alternate is the one executed under normal, error free conditions, and the acceptance test successfully passed. In case an error occurs, the secondary alternates are tried one by one, with exit from the block taking place as soon as acceptance test is passed after any of them. However, if all of them fail, the recovery block is said to have failed, and we have to go back to a recovery block at a higher level (recovery blocks can be appropriately nested). *Conversations*[RAN 75] and *fault-tolerant monitors* [KIM 79] are applications of the recovery block concept in the context of communicating processes.

2.1.5. Multiple Copy Update

A consideration of prime importance is the degree of replication of information in a distributed system. We also are concerned with their disposition on the various sites and strategies for runtime management. A large number of copies result in a small read access time and high reliability but have a high updating cost (in terms of the time required and the message traffic generated). We have investigated the problem of selecting the number of copies and their disposition assuming a simple strategy for updating them [WAH 79]. This is the problem of *file allocation* and most probably is not amenable to a polynomial time solution. [WAH 79] has developed efficient heuristics for the same problem. Many strategies for handling this problem are available ,viz. updating all the copies before allowing another update to start, updating a single copy and then relying on this copy to propagate the update to others, updating a majority of copies, etc. Each of these strategies is suitable under a different set of conditions. In case the number of data copies is small, and the read requests far outnumber the update ones, the first strategy is clearly superior. In case we have too many copies and frequent updates, the second strategy yields better results. We can think of hybrid strategies too.

2.1.6. Data Distribution

This is the task of placement of the various copies of global data at judiciously chosen locations in order to optimize some parameters like communication cost or response time. Some dynamic and adaptive scheme would most probably lead to the optimal placement, since it depends on the access request patterns. However, it would be a non-trivial job to implement such a scheme. A vast amount of literature exists in this field and a survey can be found in [WAH 79].

2.1.7. Naming and Cache Consistency Problem

There should be a systematic way for users on the network to access global information. One way is to register different pieces of global information in a name server or registration server [BIR 82]. Users can obtain the necessary information by consulting the name server without broadcasting to get it. However, resolving the name reference may incur much overhead on the application program. One solution is to cache the resolved references at local sites for later use; but then we have *cache consistency* problem. We will develop a scheme to remedy this problem efficiently.

2.2. Research Summary

2.2.1. Synchronization:

For the various nodes in a distributed system to be able to work in collusion towards a common goal, some form of synchronization mechanism is essential. The mechanism has to be such that every node in the network has roughly the same idea of time. Past research has shown that the availability of such a mechanism enables us to construct algorithms which can tackle the synchronization aspect of most of the application problems. A mechanism (called the *Global Clock Mechanism*) [GAN 84] has been developed to achieve the necessary synchronization. The *Global Clock* is a virtual clock, which is implemented using the local clocks at each node. All messages in the system are timestamped before being sent. Algorithms have been

developed to ensure that the drift between any pair of local clocks is less than some prespecified tolerance limit. Whenever a message is received its timestamp is compared to the clock value at the local host. If the value of the local clock is lesser than the timestamp, the clock is bumped forward to a value greater than the timestamp. In this way it is ensured that the various local clocks in the network will be close to each other. Algorithms have also been developed to enable ***reconfiguration***, i.e. ensuring that a node, which stopped functioning at some point in time, can reset its local clock to be in synchrony with the local clocks of other nodes.

2.2.2. Multiple Copy Update:

Fault tolerance requires that critical data be replicated and stored at more than one node in a network. However, this replication leads to certain new problems in information updating. Now care has to be taken to ensure that all the replicas of the data be consistent whenever a transaction (i.e. a ***retrieve*** or an ***update***) is to take place. Various solutions to this ***multiple-copy update*** problem have been suggested in the past. Most of them tend to be either too conservative (i.e. too careful to prevent any inconsistency in the data) and hence very slow, or not reliable enough (*performance **algorithms*** which allow for only one of the copies of the data to be updated before returning a ***done***, banking upon this copy to update the others). A technique, based on ***hot*** and ***warm*** copies of data[GAN 84], has been developed which is reliable and has a better performance than the existing schemes. Among all the sites having copies of a data item, one is designated as the ***primary*** site. The copy residing at this site is always the most recent one. It also has the responsibility of initiating updates and returning a completion message to the user process. In the database sense it can be thought of as the ***transaction coordinator***. There is a set of sites which are designated as ***hot*** sites. These sites are also the most recent ones, and form the set of sites which have to be updated by the primary before a 'done' is returned to the user process (by the primary). There is also a set of sites called the ***warm*** sites. The version of data on these is not the most recent one, rather updates are sent periodically from one of the hot sites to each

warm site. When an update is being processed, further updates have to be locked out only for the time that it takes to update all the hot sites. An additional advantage accrues if we have a lot of such reads which are not particular about the data site being absolutely current. In such a case one of the warm sites can supply the requisite information.

In the event of failure of the primary site, one of the hot sites becomes the primary automatically. When one of the hot sites crashes, one of the warm sites becomes a hot site, by getting updated. While this is going on, no more updates can be proces. ed. However when a site is becoming warm, no such locking or waiting is necessary. This technique is fault-tolerant since it has the provision of enabling a recovering node to update its copy of the data before it gets operational.

2.2.3. Consensus Problem (BGA Problem):

The *Byzantine Generals Agreement (BGA)* problem is an abstraction of the familiar problem of a number of nodes reaching consensus over the value of some datum in the presence of failures (*crashes* and *malicious* behavior of nodes). This has been found to be an exceedingly complex problem to tackle. Lot of research has been done in this area and many algorithms, both *deterministic* and *probabilistic*, have been put forward. The solutions discovered are very expensive, and hence not amenable to actual implementation. The communication overhead required in terms of the number of messages is exponential. Recent research shows that under some restricted types of failures, and making some assumptions about the behaviour of the processors and the communication medium, we can get polynomial solutions. However, these polynomials are of quite high order.

We have developed a new model for the consensus problem and call it the *Generalized Byzantine Generals Agreement (GBGA)* problem. Algorithms have been developed to solve this problem under various conditions. In the consensus scenario, we have a set of processors, each of them having a value for some datum. Each of them has to transmit its value to all the non-faulty processors. In the standard BGA algorithms, the way this is done is to let each processor having the data work as the general once. This processor

transmits its data to other processors, and they go ahead and reach consensus on its value among themselves. For the whole problem to be solved, the BGA algorithm for one general has to be run N times (if there are N processors wishing to transmit data). Our approach has been to let the consensus reaching on all the values proceed in an interleaved manner; analogous to the requirements of *strong* and *weak* consistency conditions for database updates. This approach seems to have promise as it has potential for high degree of parallelism.

However, we have not carried out any experiments so far which could give us a measure of the actual execution speed, and hence the usefulness of the algorithms developed for the GBGA model.

2.3. Future Research Tasks

In the following sections, we give more detailed discussion of some specific topics to be studied in future.

2.3.1. Collection of Global Information

The collection of status information would be much easier if we have a fully connected network, like Ethernet. A process could be assigned the task of monitoring messages passing on the network and collects interesting information, similar to the idea of *Publishing* [POW 83]. We'll call this process a *Daemon* and let it have the capability of the accepting requests of the following form: { event, action }. This is an extended idea of the *event* handler of [BAL 79]. The kind of events we would usually be interested in is the failure or overloading of nodes. If a process is interested in finding out when or whether a node fails, it can send a request to the daemon, and the daemon process will take the responsibility of watching the status of the node. Whenever that node fails, a message or mail as specified by the action may be sent to the process which initiates this request. Note that the action may be quite general; it can be an abortion request for some computation, or it may be a request to initiate a new computation at some particular node. This { event, action } pattern is quite similar to the paradigm of production system.

There are two reasons for adopting a daemon in the network:

(1) The daemon may watch the node status for different application programs so that message traffic could be reduced substantially than in the case that each application process has to periodically send messages to detect failures. Furthermore, the daemon sends messages only when it detects that a node has not been sending message for quite a while.

(2) The daemon is an expert in watching status and taking particular actions. It can be built as a separate module and thus its capability can be enhanced without modifying each individual application program.

We will look into the design issues(e.g. fault tolerance of the daemon) involved in building a daemon and to explore its applications. In large dynamic networks, there will be the problems of exchanging information among daemons and of constructing global view from the information provided by the daemons.

2.3.2. Distribution of Global Information

Different pieces of global information may be replicated and distributed all over the network. To facilitate the access of these global information, we propose to use an extended registration server and a caching scheme to provide reasonable performance. The extended registration server is capable of storing different types of registries according to the characteristics of different pieces of global information. Furthermore, we may generalize the idea to an *Object* Server, i.e. each object is an abstract data type and has its associated operations. This would provide a certain degree of security and *information* sealing. However, this is an open research problem, and many issues need to be studied.

To solve the cache updating problem, we may employ the idea of deamon process mentioned in the last section. If a process maintains a cache at a local site, it can send a request to the daemon for watching any updating activities going on, and the daemon will send a update request to the process when it detects one. In this case, the process does not have to periodically checking the currency of its cached information by itself. We will look into

the design issues involved in building such a caching scheme.

2.3.3. Multiple Copy Update:

We shall be looking into the performance, practicality and implementation of the suggested algorithms. We shall try to find some relations between the number of copies of an item of information and the time required to access it; and also the period for which the system can function (i.e. at least one copy of the information is still active). Additional complexities can arise if some nodes fail during reconfiguration, etc. These issues shall be looked into.

3. ROUTING TECHNIQUES

3.1. Introduction

Our objective is to develop distributed and reliable routing techniques for *large* and *dynamic* networks for military application. As in the actual combat field, links and nodes can fail and recover at any rate at any time, it is necessary that route must be adaptively changes and maintained as the topology of the networks changes. Thus, in order for routing algorithms to be useful in the battlefield, routing algorithms must be extremely reliable, i.e. it must be able to adapt any arbitrary changes in network topology at any arbitrary rate.

Our research thus follows three main steps: (1) distributed routing algorithm, (2) hierarchical routing algorithm of fixed clustering structure, and (3) finally directory management to maintain and update the adaptive clustering structure. We first investigated the distributed routing techniques, since they could be used in a dynamic environment. They could automatically change the route as the network change the topology, and they are much more reliable than the centralized counterparts.

As the target network is assumed to be very large, conventional nonhierarchical routing scheme could not be used. This is due to that fact, the communicational overhead increase dramatically as network become very large. We thus investigated the adaptive hierarchical routing algorithms for hierarchical networks with fixed clustering structure, i.e. the clustering structures of the networks do not change with time. The main criteria of developing hierarchical routing schemes are that they must be reliable too, as in the nonhierarchical case.

Finally since the target network is assumed to be mobile, the network topology keeps on constantly changing, it is necessary from time to time to change the clustering structure. Furthermore, it is necessary that the resulting clustering structures are balanced, i.e. there are more or less same number of nodes in each cluster. If the cluster is either too small or too large, the overhead would be excessive. The clustering procedure should be reliable too.

3.2. Past Achievements

Classification of Distributed Routing Techniques

We have proposed a classification scheme for distributed routing algorithms based on data structures stored at each node. Most of the distributed routing algorithms could be classified by this classification. This classification gives the performance upper limit for each type of distributed routing schemes. Thus, by using this classification, we could determine the best performance that each distributed routing algorithm could achieve. We have used this classification to evaluate many distributed routing protocols proposed in the literature. Preliminary results have been reported in [TSA 82], and currently a paper is now in preparation.

An Adaptive Hierarchical Routing Algorithm

We have proposed an adaptive hierarchical routing scheme which has many desirable properties [RAM 82, RAM 83]: it is distributed, is very reliable, and works for any arbitrary hierarchical network of any levels. It uses many routing controllers so that failure of any one of them would not halt the operation of the network. The algorithm could reconfigure rapidly as network changes the topology. The communicational and computational overheads are also minimized. The proposed scheme is more robust than Baratz's scheme [BAR 83]. It also does not have the loop problem as in [MCQ 74, KAM 76].

Classification of Adaptive Hierarchical Routing Techniques

Based on the classification of distributed routing algorithms, we have further proposed a classification scheme for hierarchical routing protocols. There are around 700 adaptive hierarchical routing schemes have been identified under this classification for a 2-level hierarchical network [GRA 83, RAM 84c]. This classification also gives the performance upper limit for each type of hierarchical routing scheme. The proposed classification is better than the previously proposed scheme [HAG 83], where only static hierarchical routing scheme could be classified.

3.3. Current Research Progress

Directory Management

In large and dynamic networks, it is necessary to partition networks into clusters in order for efficient control and management. Clustering of networks has to be done in real time in order to response to the network status change. There are several ways to do clustering: centralized, distributed and hierarchical approaches. Centralized method first select a controller in each network, after collecting all the topological information from all other nodes, the controller then computes to obtain the best clustering structure. The main disadvantage of this approach is that it is not reliable. Distributed approach requires all nodes in networks to participate in order to obtain the clustering structure. The main disadvantage of this approach is that it may take too much time to obtain the clustering structure and the clustering structures obtained are not necessary optimal. However, hierarchical approach could have advantages of both centralized and distributed approaches without their advantages. Thus, hierarchical approach will be pursued.

In order to obtain good clustering structures, we propose to cluster networks as distributed B-tree's. A B-Tree is a tree with all leaves distributed uniformly throughout the tree, and there is no side that has very large number of leaves attached to it, while the other sides have only few leaves. If networks are partitioned as distributed B-tree's, each cluster will have more or less same number of nodes within it. Networks with balanced clustering are easier to manage and give better performance than networks with unbalanced clusters. Since each cluster has more or less same number of nodes, it is also easy to carry out reconfiguration if necessary.

Currently, we are designing various strategies, including hierarchical centralized and hierarchical distributed methods, to cluster networks into distributed B-tree's [RAM 84e].

Simulation

The simulators for the original Arpanet, the new Arpanet and the proposed adaptive hierarchical routing algorithms have been successfully completed and documented this year [RAM 84a, RAM 84b]. Currently, we are collecting data and interpreting the result [RAM 84d].

3.4. Future Research Tasks

Directory Management

This task has just formulated this year with preliminary results [RAM 84e], we could see a lot work ahead.

Optimal Clustering Structure

Given a network topology, traffic requirement and a hierarchical routing algorithm, it is necessary to determine what is the optimal clustering structure for operation. The clustering structure will affect the performance greatly. This can be illustrated by observing the effect of changing the size of clusters of a 2-level networks. If the size of each cluster is one, then there is no clustering effect. However, if the cluster is large enough to include all the nodes in it, there is no clustering effect either.

We will cluster the networks so that the reliability would be optimized, and communication and computational overhead would be minimized.

4. COMMUNICATION PROTOCOL SYNTHESIS

4.1. Introduction

Protocol synthesis is a process of designing new communications protocols. The objective of developing automatic protocol synthesizer is to provide a systematic way of designing protocols such that their correctness can be ensured. Although protocol analysis methods are useful to various extents in validating existing protocols, they do not provide enough guidelines for designing new protocols. What designers need in designing new protocols is some set of design rules or necessary and sufficient conditions to follow so that the protocols designed are guaranteed to be correct. Then, the newly designed protocols need not go through the analysis stage to be checked for their correctness.

4.2. Past Achievements

We have developed a systematic protocol synthesis procedure which construct the peer entity from the given local entity which is modeled by a Petri net.[Do83] If the given entity model satisfies certain specified constraints, the protocol generated will possess those general logical properties which are what a protocol synthesizer is looking for. The synthesis procedure is very general. It is applicable to every layer of the protocol structure.

To construct the desired peer entity model, there are three tasks which should be conducted in sequence :

(1) Check local properties of the given local entity model to make sure that it is well-behaved. This can be done by generating and examining the structure of its state transition graph.

(2) Construct the peer state transition graph from the above generated state transition graph according to some well designed transformation rules.

(3) Construct the peer entity model in Petri nets from the peer state transition graph.

The protocols generated are guaranteed to be logically correct if the given entity model satisfies certain desirable local properties. These desirable local properties can also serve as guidelines in designing the given entity model.

We have the following complete list of desirable local properties for the given entity:

(1) Local completeness;

(2) Local boundedness;

(3) Local liveness;

(4) No undesirable terminal states in STG1;

(5) No cycles of send transition;

(6) No cycles of receive transitions; and

(7) Every reachable state in STG1 can reach at least one of the desirable terminal states.

Also APS (Automated Protocol Synthesizer) has been implemented as a computer aided design tool on Vax 11/780 machine. The APS was programmed in C language. The code size is 3.5 k lines long and occupies 20k byte memory. It can accept a given entity model up to 50 places and 50 transitions. However, links between places and transitions are dynamically allocated and hence there is no restriction placed on the maximum number of links. Therefore, at the current version of APS, communication protocols with moderate complexity can be adequately handled.

4.3. Future Research Tasks

(1) State Explosion Problem

The synthesis of complex communication protocols by using transition oriented model is faced with state explosion problem. This problem not only complicates the validation of general protocol properties (such as deadlock-freeness and proper termination) but also imposes further difficulties on the analysis of performance and data transfer aspects of communication protocol (such as timers and sequence numbers). Because Petri nets are a

sort of transition oriented model, a resolution of state explosion problem is urgently required. Since the complexity of communication protocol is proportional to the number of reachable states, fundamental principle in managing the complexity of protocol synthesis is to reduce the size of the reachable state space. The Petri net abstraction technique has been developed to achieve this objective. It originated from two sources, top-down design methodology and hierarchical modeling capability of Petri nets.

The above technique becomes feasible in our APS because of hierarchical modeling capability of Petri nets. In the Petri nets, an entire subnet can be replaced by a single place or a single transition for modeling at a more abstract model. On the other hand, we may also replace a place or transition with a subnet to provide more detailed modeling. The former procedure is called abstraction and latter elaboration. Abstraction is good for system analysis because it simplifies the model. However, elaboration is beneficial to system design because the system design may start from a simple and easily understood model. In order to insure that local properties still be analyzable from reduced Petri nets, the abstraction cannot be performed arbitrarily. The abstraction becomes meaningful only if the desirable protocol properties are retained in reduced Petri nets. In designing local entity model, we locate as many well-behaved modules as possible so that their replacement with single transitions simplify the design procedure preserving certain desirable properties of communication protocols. This procedure can be applied recursively to reduced Petri nets themselves, and hence the overall complexity of protocol synthesis can be substantially reduced.

(2) Error Recoverable Protocol Synthesis

The development of automated protocol synthesizer is a rapidly growing subject in many applications. However, in our approach, it was assumed that the communication environment is perfect. The protocol synthesis procedure can be extended to cover the circumstances associated with unreliable links. Error recovery strategies such as time-out mechanisms, error detection coding schemes and time-stamp mechanisms will be studied to incorporate those strategies in entity models for handling various types of communication errors.

(3) Performance Evaluation of Communication Protocols

In order to study the performance of a protocol some indication of elapsed time or execution time must be added to the formal protocol specification. Once time is included in the specification, several paths open for future automated exploration. Among the most important of these potential paths to explore are:

(a) analysis of the protocol to check for logical correctness of the protocol including the time specifications,

(b) simulation to predict performance, and

(c) analysis to predict performance.

(a) Analysis including time specification

Analysis has traditionally been done considering only the sequence of events and not their duration. Adding the dimension of time duration means that certain sequences in the protocol may no longer be possible; some timeouts may never be executable; deadlocks may occur. Analysis including time specification requires an examination of such errors of logical correctness in the protocol. This topic is probably the most difficult of the three.

(b) Simulation to predict performance

Simulation is a technique in which a machine-executable model of the protocol is derived, the model is executed, and statistical records of its performance made. Interpretation of these statistics leads to an estimate of the performance. Usually, the machine-executable model is derived manually from the description of a protocol, the possibility arises of automatically producing a machine-executable model for simulation of performance.

(c) Automated analysis to predict performance

Instead of using simulation to predict protocol performance, the technique proposed here is direct analysis of the protocol specification.

5. CONTROL SCHEMES FOR PROCESS ALLOCATION

5.1. Introduction

A distributed computing system comprises a set of interconnected processing nodes that support various distributed computations. A distributed computation is defined as multiple processes working collectively towards a common goal. These processes interact with system objects distributed over the nodes, and they also communicate with each other. Overhead is incurred due to their consumption of system resources, such as processor time and communication bandwidth. In particular, if a lot of inter-node data communication is required, the communication overhead may impose a significant performance penalty on the distributed computation.

We will address the problem of process allocation in a computer network environment. In such an environment, distributed processes share the system resources, such as communication bandwidth, computation power, and data objects. These processes are created asynchronously at their *birth* nodes, and they can be allocated to remote nodes for execution. Process allocation is an optimization problem with multiple objectives including communication overhead reduction and load balancing. This problem does not exist in conventional single processor systems, whereas in multiple computer systems, process allocation is a key factor to the efficient utilization of distributed system resources.

5.2. Past Achievements

The general problem of process allocation is NP-complete. Therefore, optimal allocation is infeasible except for certain special cases. We discovered that when the interprocess communication graph is a tree, an optimal allocation can be achieved with polynomial time complexity via a dynamic programming algorithm. For the general case, however, we have to rely on heuristics that provide suboptimal allocations. An iterative improvement heuristic is devised and its simulation experiments are encouraging.

The distributed system configuration may be dynamically changing due to the fluctuating workload and migrating objects. Furthermore, the runtime behavior of a distributed computation may not be completely known *a priori*. Therefore, the effectiveness of initial allocation may depreciate to such a point that it becomes desirable to relocate the processes. We have devised dynamic allocation schemes which employ distributed versions of the aforementioned iterative improvement heuristics. They involve the coordination of nodes on which the component processes of a distributed computation are running. The synchronization scheme may allow only one node at a time to improve the allocation of its processes, or it may allow multiple nodes to proceed in parallel. Various sequential and parallel schemes are devised and analyzed.

5.3. Research Directions

We have studied various process allocation algorithms for distributed systems. Most solutions to the process allocation problem require global information about the system configuration. In a distributed computing system, all the information concerning remote nodes may not be locally available. It may involve tremendous control overhead to collect necessary information from nodes located at the far end of the network. Therefore it is desirable to adopt allocation schemes that would use partial information initially and collect necessary information as the computation proceeds.

We have devised both static and dynamic process allocation schemes for distributed system environments. In our approach, the process allocation will consist of two phases: initialization and improvement. During the initialization phase, the processes are tentatively allocated using only the information available to the controllers. During the improvement phase, the allocation controllers collaborate so as to attain a mutually agreed allocation that reduces the communication overhead as much as possible. Generally the improvement phase is an iterative one. In every iteration, one or more processes will be relocated, resulting in an improved configuration as the basis for the next iteration. Therefore the objective function value - e.g. communication overhead - will monotonically improve until a local optimum

is reached. This strategy stems conceptually from a variety of heuristics used to solve NP-complete problems.

In order to provide a pragmatic approach for real-time process allocation, various control and synchronization schemes will be evaluated. The criteria used for comparison will be the communication overhead incurred and the total execution time, including processing and queuing delays. The execution time is dependent upon the processor load conditions. In order to minimize the total execution time, load balancing techniques will also be examined.

6. LOAD SHARING IN DISTRIBUTED SYSTEMS

6.1. Introduction

In a general-purpose distributed system, it is desirable to automate the allocation of processor resources across the network. Load-sharing is a policy for redistributing the aggregate demand among the processors in the network so that the overall performance of the system improves. In this section, we discuss the general problem of load balancing, and discuss the feasibility of migrating processes besides process allocation in a network as already discussed in the last section.

Optimal load-sharing requires foreknowledge of the runtime characteristics of all processes and is computationally expensive. A suboptimal heuristic algorithm allocates in real time according to the currently observed state. The fundamental question in devising a heuristic load sharing policy is the dimension of the decision state. Policies that use only information about the average behavior of the system, ignoring the current state , are termed static policies. Policies that react to current state are termed dynamic policies. Static policies use no state information, hence are simple, stable, but suboptimal in the sense that they do not react to the current state of system. On the other hand, dynamic policies have to constantly update their state information. This arises the overhead problem in maintaining the state information up-to-date and making wrong decisions based on out-of-date information. This tradeoff is in the heart of our research in finding a simple, stable load sharing policy that takes minimum but meaningful current state into consideration in balancing the load across the network.

6.2. Past Accomplishments

The mean response time can be improved by redistributing the total workload among processors by migrating processes from heavily loaded processors to lightly loaded ones. A process that migrates from one processor (the source) to another (the destination) incurs relocation and communication cost. A migrating process incurs burden of address fixing

and message delivery on the originating processor. A migrating process has to contend for the access to the communication network; hence, incurring the communication cost.

The execution time for each process is assumed to be exponentially distributed with the parameter depending on the executing processor. Relocation and communication delay are also assumed to be exponentially distributed. Then the problem is formulated as a static (probabilistic) control policy for a set of queues to minimize the average response time. The optimal rate of local and remote processing at each processor (in static sense) is obtained.

Since the arrival rates at each processor may change in time, a simple estimator for intensity function of the arrival process is derived. This estimator determines how often the static optimization must be carried out.

Once the migration of a process has been sought, the load sharing algorithm must choose which process should migrate from among those resident on the overloaded processor. Factors that are important in choosing the best process to migrate will also be discussed.

6.3. Research Directions

We are studying various heuristic threshold policies. These policies are simple, and use state information that their rate of change is such that every node can have a correct view of the state at every node in the network.

The definition of load in all the published work in this area is based on the number of ready runnable processes. This metric for the load measurement is simplistic. On the other hand, load is a fuzzy parameter. We intend to study the load sharing problem in the context of fuzzy logic and approximate reasoning. Fuzzy logic provides simple solutions to complex problems where the range of variables is fuzzy.

7. COORDINATION OF DISTRIBUTED COMPUTATION

7.1. Introduction

In the former sections, we discussed important issues in the control and management of a large distributed system. Now, it has come to the point to utilize these mechanisms as a underlying basis for real application programs in a distributed environment. However, to support distributed computations, we must consider the problem of coordination among those nodes that participate in the computation. An effective strategy of coordination of the distributed computation may achieve high resource utilization and the desired degree of fault tolerance in the network. In the context of large dynamic networks, this would help save some precious computation results in a dynamically changing environment and satisfy the goal of real time processing.

Since a large job in dynamic networks may require complex interactions among different nodes, we propose to extend the idea of *Make* [Fel 78] program to the case of distributed systems and use a *Makefile* to specify the necessary interactions and job steps among working processes. The idea of a distributed *Make* program is to reduce the amount of effort required in keeping track of what changes have been made by various people, working together in a distributed environment on the same task, and repeat only that part of a computation which is affected by the changes. There are three potential advantages of using such a distributed *Make* program:

(1) Elimination of Redundant Job Steps. A user may not know that another user has performed some computations at a remote site but a *Make* program can find that out and avoid redundant computations.

(2) Potential Load Balancing. In a network, there are usually some nodes sitting idle or lightly loaded. If a job requires substantial amount of computation, the load can be distributed to a set of nodes, and the *Make* program could take care of the coordination task among these nodes.

(3) Fault Tolerance. For large jobs, we would like to save the computation results even when some nodes which participate in the computation fail.

The way to recover a computation can be specified either in the *Makefile* or the *Make* program can take care of some default recovery action. This idea would be elaborated later in this section.

7.2. Research Directions

To use the *Make* program, a user has to specify the computation steps and dependent files necessary to make a target file in the *Makefile*. The specification language used to build the *Makefile* limits the capability of the *Make* program. We will study this problem further later on. For the present, let us examine a simple *Makefile* and use this example to illustrate the interactions among the set of computing components. This following *Makefile* is used to specify a computation which makes a book; chapter 1 has some graphs, chapter 2 has some tables and equations, and chapter 3 has some graphs and equations. Therefore, corresponding filters are used to preprocess individual chapters:

```
book: ch1.g ch2.te ch3.ge
        ditroff -Pip -ms -t ch1.g ch2.te ch3.ge > book
        lpr -Pip -n book
ch1.gr: ch1 f1.g f2.g
        grn† -Pip ch1 > ch1.g
ch2.te: ch2
        tbl ch2 | eqn > ch2.te
ch3.ge: ch3 f3.g
        grn -Pip | eqn > ch3.ge
```

The data dependence graph and control dependence graph derived from this file are shown in Fig. 1a and Fig. 1b. To facilitate the discussion, we define the following terms which are used in our model:

Coordinator: the process which runs the *Make* program and coordinates the computation among participating nodes.

† *Grn* is a filter used to process line drawing commands created by a graphical package(developed at Berkeley) called *Gremlin*.

Worker: a process which participates in the computation.

Daemon: a process that monitors the status changes of the coordinators and participants of the computation as mentioned in section 2.

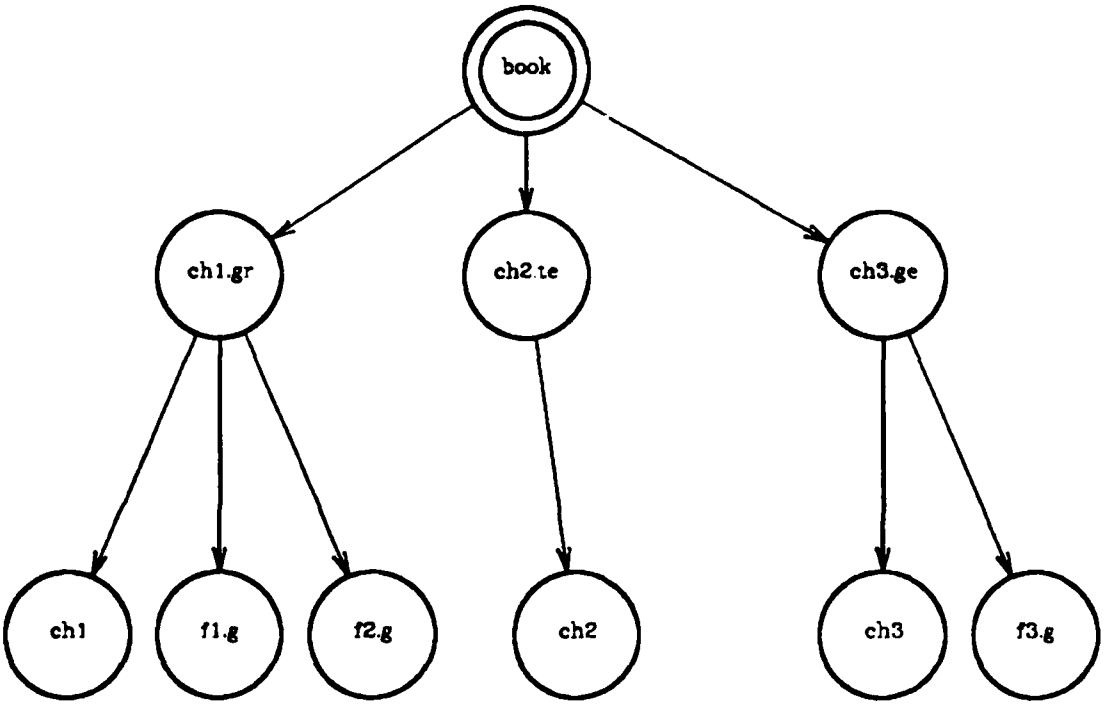


Fig. 1a Data Dependence Graph of the *Makefile*

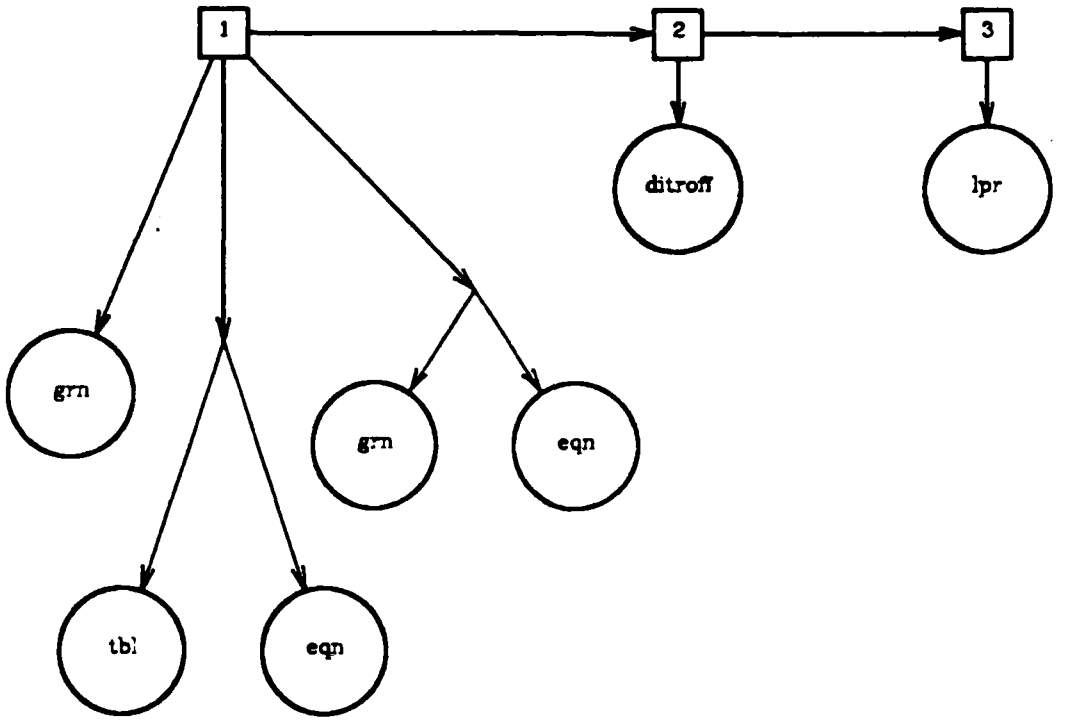


Fig. 1b Control Dependence Graph of the *Makefile*

A job is defined as a tree of computation as can be seen in Fig. 1b. Note that square nodes connected by arrows specify sequential control dependence, while a fork of two nodes specify the *pipeline* dependence; otherwise, the processes are independent. As we can see from the above example, the two computational steps (ditroff and lpr) of **book** has serially dependent, and the two computational steps of **ch2.te** are pipelined. The book depends on three files: **ch1.g**, **ch2.te**, and **ch3.ge**, and **ch1.gr** depends on **ch1**, **f1.g**, **f2.g**, and so on. **Ch1.gr**, **ch2.te** and **ch3.ge** may be independently processed, and later on, they may be combined together to form the book. We see that the *Makefile* provides us with sufficient information about the control dependence and data dependence. Therefore, we could use this information to decide whether two processes can be distributed to different nodes or not. However, the current *Makefile* format is not sufficient for our purpose in specifying all requirements of distributed computations; e.g. for some particular application, we may want to restrict the distribution of computation to a particular subset of nodes. We would like to develop in the

future a new language to be used in specifying distributed computation in *Makefile*.

The *Make* program(i.e., the *Coordinator*) starts by examing the *Makefile* and interacts with the file server to get the update time of all related files and decide what job steps have to take. The coordinator then select nodes to participate in the computation according to the constraints imposed in the *Makefile*‡. Then, a working request is sent to each participating node to create a *worker* process and a corresponding request is sent to the daemon server to ask for watching the status changes of that node. After that, the coordinator waits for two things:

(1) Message of completion or results from the workers. The coordinator may initiate another computation step at this time if allowed according to the control dependence graph.

(2) Message from the deamon regarding the failure of some worker. In that case, the coordinator can select another node to initiate a new worker process or abort the whole computation according to the specification of the *Makefile*.

Many issues are involved in building a distributed *Make* program:

- What is the right way to specify data dependence and control dependence in the *Makefile*? Is the scheme used in Unix sufficient? If Unix scheme is used, what assumptions regarding the file servers have to be made?

- What is the right strategy to select nodes to participate in a particular computation? Note that depending on the semantics of a command in the *Makefile*, there will be certain restrictions imposed on the node selection. For example, user may find it necessary to execute certain commands at particular nodes because of location-dependence within the command. Even if the user places no restriction on selection of

‡ The selection algorithm may it self be a research problem in load balancing as discusses in section 5 and section 6.

nodes, the *Make* program may have to introduce some restrictions so that heterogeneity of nodes doesn't cause a problem.

— How do you handle the problem of node failures? Should we abort the computation or select a new node to continue the processing? Could a user specify in his *Makefile* about what he wants to do when some node fails?

— How is it decided by the *Make* program that a file to be used in the computation has been updated since last use? This problem may be complicated if the files are replicated and clocks of various file servers are not synchronized.

We intend to investigate the above issues and find the answers in this project.

8. INTELLIGENT CONTROL IN LOCAL DISTRIBUTED ENVIRONMENT

8.1. Introduction

Both database (DB) and artificial intelligence (AI) systems must represent and process knowledge about the real world. Although most of the researches in these areas have been conducted along their own lines, they are essentially complementary in the sense that both fields have a great deal to contribute to each other: DB has more practical experience in security, efficiency, and reliability; AI has developed more sophisticated techniques for representing the meaning of data and solving complex problems in specific task domains, e.g. expert systems.

In this research we are interested in the application of both AI and database technologies to a specific domain, dynamic computer networks. Across a large spectrum of interesting subjects related to this area we are attacking three major problems:

a) Communication Subnet Interconnections: Assume each node in the network has limited capacity in terms of computation as well as communication loads (say, 1 for computation and d for communication; i.e. at any time, at most one computation can be running and at most d communication flows are allowed to pass through at any node). The communication subnet interconnection problem can be formulated as follows: Given the current configuration of the network and a specification of net(s) to be connected, how can we conduct the interconnections such that the specification is satisfied and none of the nodes in the network is overloaded (in terms of either computation and communication). Furthermore, once the subnets are interconnected, the connectivity of them should be maintained subject to the dynamics of the system.

b) Query Processing: One of the major problems of distributed systems is database. Basic database operations include selection, projection, and join. In distributed systems query processing is complicated by the communication cost due to the data transfer among nodes. In the context of local dynamic networks, query processing is different from

that of ordinary distributed systems for

(i) Concurrent processing is feasible

(ii) Communication links can be set up more flexibly and therefore affects the communication cost involved

(iii) The interconnection problem underlied

c) Control: As the requests for subnet interconnections and query processing may arise continuously, certain control to resolve resource conflicts and to assure the global performance of system is necessary. Again, the control problem is seriously affected by the dynamics of the system.

8.2. Past Achievement

Most of the related work in the area of dynamic communication networks were conducted in the subjects of routing [RAM 83] and hierarchicalization [RAM 84a]. The proposed communication subnet interconnection problem differs from the routing problem in the following sense:

a) Routing problem focuses on the point to point (i.e. only two points are involved) communications issues while the subnet interconnection problem concerns mostly multi-point communication problems.

b) The subnet interconnection problem regards net maintenance as an integral part of the problem.

On the other hand, unfortunately, no query processing or control issues have been seriously considered in the literature.

8.3. Current Research Progress

The work been conducted during the report period can be summarized as follows:

a) Representation and Modeling Techniques: The network state can be directly modeled by two proposed constructs: the access graph, where each node is a vertex and two nodes are connected by an edge if they have direct communication link, and the availibility graph, where two

nodes are connected by an edge if there is some available communication capacity between them.

b) Subnet Construction Problem: The subnet construction problems can be classified into three categories, according to the specification given:

(i) It has only two nodes, or

(ii) It has multiple(more than 2) nodes, or

(iii) It has a set of multiple nodes(2 or more).

Case(i) can be solved easily by the shortest path algorithm. Case (ii) can be reduced to the Steiner tree problem and is proved to be NP-complete. Case (iii) is found to be similar (but more complicated) to the circuit routing problem and unfortunately is again an NP-complete problem. Heuristics taking into account load balancing for case (ii) and case(iii) have been developed. Emphasis have also been placed on the following issues:

(i) If all of the heuristics fail to interconnect the nodes involved, how to disturb the least number of existing nets such that those disturbed nets can be re-interconnected to accommodate the new net(s).

(ii) Rules about congestion removements, redundancy removements and reinterconnections.

c) Query Processing: A parallel pipelined multi-relation join algorithm has been developed and proven to be better than any of the existing methods. Tradeoffs between communication and computation are also considered. The relationship between the computation issues and the underlying interconnection issues have also been considered.

d) Control: Algorithms to schedule communication requests are developed. All requests are regarded as competing goals and can be classified into accomplished goals and pending (unsuccessful) goals. Movements of goals between classes due to events (e.g. node/link addition/deletion, net removements) are dynamic. Procedures for forward reasoning (event and state driven) and partial result saving are developed. To improve the performance, preemption is introduced and an algorithm based on backward reasoning which creates least perturbation is derived. Meta-

control issues like resource relocation are also considered.

e) Knowledge base and Database support: Since the algorithms (heuristics, experts) introduced above access high order objects such as access graph, access graph with net labeling, it will be wise to place these objects in the database. However, instead of storing the objects directly, definition(knowledge) of the objects can be stored and objects can be derived dynamically. Nevertheless, the objects may be required to be stored explicitly if the access frequency is high. Guidelines to make such decisions are developed. Currently, deductive knowledge about the high order objects based on the first order logic have been developed. On the other hand, active forward knowledge(rules) about maintaining the objects are also completed. Methods to deduce automatically the active rules from deductive knowledge are also developed

8.4. Future Research Tasks

Our future work along this direction will be conducted in the following issues:

1) network state estimation without continuous state reporting. Specifically, we are interested in deriving the network state like relative locations based on partial knowledge, for instance, neighboring events,

2) performance evaluation and comparison among the algorithms developed,

3) other meta control mechanisms in the control area, for instance, scheduling function discrimination.

9. SUMMARY

Our proposal on the design and management of large, dynamic, and unreliable distributed networks can be summarized as follows:

(i) In the area of *global information management*, we will

a) develop appropriate algorithms for the problems of synchronization, multiple-update, data collection, accessing of global information and currency in the dynamic environment.

b) investigate the impact of radio communication

c) improve the Byzantine agreement algorithm by distinguishing read and update accesses.

d) study the problems of hierarchicalizing, which include the choice of number of levels, the allocation of positions in the hierarchy to nodes in the network, etc.

(ii) In the area of *routing control*, we will study the following issues:

a) distributed clustering algorithms

b) classification and comparison of distributed, hierarchical routing algorithms

c) routing information update protocols

d) directory management

e) simulation and performance evaluation

(iii) In the area of *communication protocol analysis/synthesis*, we will study the following issues:

a) state explosion problem

b) error recoverable protocol synthesis

c) performance evaluation of communication protocols

(iv) In the area of ***process allocation***, we will study the following issues:

a) static and dynamic process allocation schemes

b) evaluation of various control and synchronization schemes

(v) In the area of ***load sharing***, we will study the following issues:

a) static and dynamic load sharing policies

b) various heuristic threshold policies

c) the load sharing problem in the context of fuzzy logic and approximate reasoning

(vi) In the area of ***coordination of distributed computation***, we will study the following issues:

a) design of a distributed ***Make*** program

b) the usage of daemon in a distributed system

c) the design of a specification language for distributed computation

(vii) In the area of ***intelligent control***, we will study the following issues:

a) representation and modeling techniques

b) subnet construction problems

c) query processing

d) control algorithms

e) knowledge base and database support

f) alternative cellular structures under which performance can be optimized

g) network state estimation without continuous state reporting

h) performance evaluation and comparison among the algorithms developed

i) other meta control mechanisms in the control area

REFERENCES

[BAR 83] Baratz, A., et al., "Establishing Virtual Circuits in Large Computer Networks," Proc. of INFOCOM 83, San Diego, CA, April 1983.

[BER 81] Bernstein, Philip A.,et. al., "Concurrency Control in Distributed Database Systems," Computing Surveys, Vol.13, No.2, June 1981.

[BIR 82] Andrew D. Birrel, et. al. "Grapevine: An Exercise in Distributed Computing." Communications of the ACM, Vol. 25, No. 4, April 1982, pp. 260-274.

[DON 83] Dong, S.T., "The Modeling, Analysis and Synthesis of Communication Protocols", Ph.D. Dissertation, Dep't of EECS, U.C. Berkeley, 1983.

[FEL 78] S. I. Feldman, Unix Programmers' Manual, Vol. 2, Bell Laboratories, Murray Hill, New Jersey 07904, August, 1978

[FIS 83] Fischer, Michael J., "The Consensus Problem in Unreliable Distributed Systems: A Survey," YALEU/DCS/RR-273, June 1983.

[GAN 84] Ganesh, Shivaji L., Ph.D. Dissertation, University of California, Berkeley, Jan. 1984.

[GRA 83] Graff, C., et al., "Control Functions in Distributed Systems," Proc. of IEEE International Workshop on Computer Systems Organization, New Orleans, Louisiana, March 1983.

[HAG 83] Hagouel, J., "Issues in Routing for Large and Dynamic Networks," Ph.D Thesis, Dept. of Electrical Engineering, Columbia University, April 1983.

[KAM 76] Kamoun, F., "Design Considerations for Large Computer Communication Networks," UCLA-ENG-7642, 1976.

[KIM 79] Kim K.H., "Error Detection, Reconfiguration and Recovery in Distributed Processing Systems," Proc. 1st Int'l Conf. on Distributed Computing Systems, Oct. 1979.

[KOH 81] Kohler, Walter H., "A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems," Computing Surveys, Vol.13, No.2, June 1981.

[LAM 78] Lamport, Leslie, "Time, Clocks, and the Ordering of Events in a Distributed System"' CACM, Vol.21, No.7, July 1978.

[LAM 80] Lamport, L., R. Shostak and M. Pease, "Reaching Agreement in the Presence of Faults," JACM, Vol.27, No.2, April 1980.

[LAM 84] Lamport, Leslie, "Using Time instead of Timeout for Fault-Tolerant Distributed Systems," ACM Trans. on Prog Langs, Vol.6, No.2, April 1984.

[MCQ 74] McQuillan, J., "Adaptive Routing Algorithm for Distributed Computer Networks," Ph.D Thesis, Harvard University, Cambridge, MA, May 1974.

[POW 83] Michael L. Powell and David L. Prestto, "Publishing: A Reliable Broadcast Communication Mechanism." Proceedings of the 9th SOSP, Operating Systems Review, Vol. 17, No. 5, November 1983, pp. 100-109.

[RAB 83] Rabin, M.O. "Randomized Byzantine Generals," Harvard University research report, 1983.

[RAM 82a] Ramamoorthy, C.V. and S.L.Ganesh, "Global Information Management," UCLA Packet Radio Analytical Workshop, Aug. 1982.

[RAM 82b] Ramamoorthy, C.V., and Tsai, W.-T., "Update Protocols for Hierarchical Routing Algorithms," Proc. of Packet Radio Analytical Workshop, UCLA, Aug. 1982.

[RAM 83] Ramamoorthy, C.V., and Tsai, W.-T., "An Adaptive Hierarchical Routing Algorithm," Proc. of Compsac, Nov. 1983.

[RAM 84a] Ramamoorthy, C.V., Nishiguchi, O., and Tsai, W.-T., "Simulation of Hierarchical Routing Algorithms," Report No. UCB/CSD 84/185, Computer Science Division, University of California, Berkeley, California 94720.

[RAM 84b] Ramamoorthy, C.V., Nishiguchi, O., and Tsai, W.-T., "Simulation Programs of Routing Algorithms," Computer Science Division, University of California, Berkeley, California 94720.

[RAM 84c] Ramamoorthy, C.V., and Tsai, W.-T., "Update Protocols for Hierarchical Networks," in preparation, Computer Science Division, University of California, Berkeley, California 94720.

[RAM 84d] Ramamoorthy, C.V., Nishiguchi, O., and Tsai, W.-T., "Performance Evaluation of Hierarchical Routing Algorithms," in preparation, Computer Science Division, University of California, Berkeley, California 94720.

[RAM 84e] Ramamoorthy, C.V., et al., "A Directory Management Algorithm for Dynamic Networks," submitted for publication, 1984.

[RAN 75] Randell, B., "System Structure for Software Fault Tolerance," IEEE Trans. on Software Engineering, June 1975, pp. 220-232.

[STO 83] Stockmeyer, L., D. Dolev and C. Dwork, "On the Minimal Synchronism Needed for Distributed Consensus," Stanford University research report, 1983.

[TSA 82] Tsai, W.-T., "Routing Techniques for Dynamic Computer Networks," M.S. Report, Computer Science Division, University of California, Berkeley, CA 94720, Aug. 1982.

[WAH 79] Wah, B.W., "A Systematic Approach to the Management of Data in Distributed Databases," Ph.D. Dissertation, University of California, Berkeley, Aug. 1982.

APPENDIX — Thesis Written Under BMD Project

Following is a list of all Ph.D. Dissertations written under BMD project. A copy of each dissertation is attached at the end of this report.

[MA 81] Ma, Y.W., "Techniques for Design and Management of Dynamic Computer Networks", Ph.D. Dissertation, University of California, Berkeley, November 1981.

[DON 83] Dong, S.T., "The Modeling, Analysis and Synthesis of Communication Protocols", Ph.D. Dissertation, University of California, Berkeley, April 1983.

[GAN 84] Ganesh, Shivaji L., "Availability and Consistency of Global Information in Computer Networks", Ph.D. Dissertation, University of California, Berkeley, Jan. 1984.

DTIC

FILMED

4-86

END